DATA ANALYSIS OF A DOCUMENT TRACKER

A Python-Based Application

NOVEMBER 28, 2018

NISHNA AJMAL

Introduction

The project aims to develop a Data Analysis of a Document Tracker application in Python. The software should perform functionalities like

- View by countries
- View by Continent
- Find the most popular browser
- Produce a list of 10 also-like documents
- Plot digraph of the also_like documents
- GUI to process the required task

"Python is a dynamic and flexible language. Python design patterns are a great way of harnessing its vast potential.". (Boyanov, 2018)

The project implements Strategy pattern using the principle of single responsibility in the modules 1 to 4.

As part of this design, the modules in the project would follow along with a combination of OOP and functional programming using the advanced language constructs in Python like

- Higher order functions -map,reduce
- Dictionary Comprehension,
- List comprehension
- Classes- Method Overloading
- Lambda expressions within higher order functions
- Regular Expressions
- Single Responsibility Principle.

The application would be developed in Spyder 3.2.8 using Python 3.6.5 64bits on Windows.

Requirements Checklist

No	Functionalities	Status
1	Implement Core Logic in Python	Achieved
2	Views by country	Achieved
3	Views by continent	Achieved
4	Views by user_agent	Achieved
5	Views by browser	Achieved
6	Implement a function that returns the list of readers of a document.	Achieved
7	Implement a function that returns the list of documents read by a visitor.	Achieved
8	Implement a function for the "also like" functionality	Achieved
9	Generate a graph that displays the "also like" functionality	Achieved

Design Considerations

Design Pattern

Chain of Responsibility pattern simplifies the interconnections between objects. Rather than maintaining references to all candidate receivers by senders and receivers, each sender holds a single reference to the head of the chain, and the rest would hold a single reference to their immediate successor in the chain. (Sourcemaking.com, 2018)

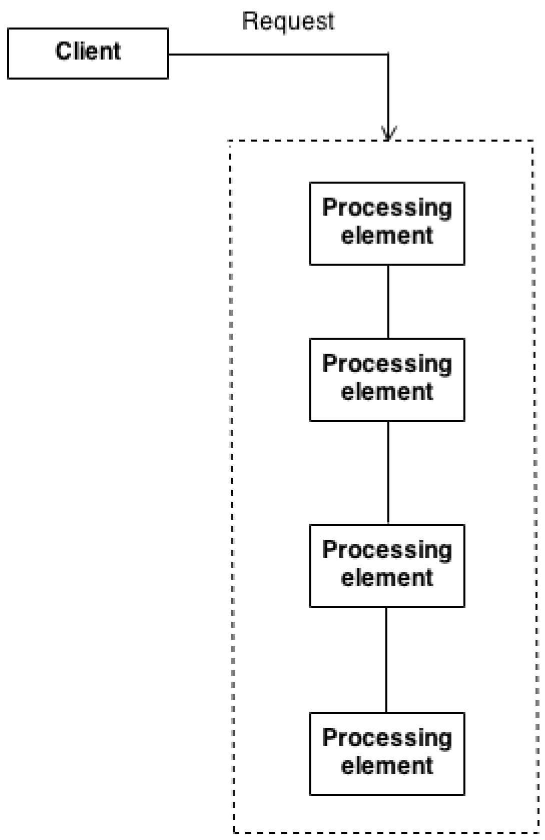


Fig : **Chain of responsibility** (Sourcemaking.com, 2018)

The pattern chains the objects which then passes any request messages from object to object until it reaches the one capable of handling the message.

Class Design

The application follows a modular structure. The different modules for various functionalities and its respective class design are given below.

Module – Module-Common Class - common_functions

a. The function **read_data** reads the JSON data in the input file path as pandas data frame

b. The **get_values function** returns a filtered list of values from specified column input that takes as input 1. A variable which has the column to be matched with value 2. Value of the selection criteria 3. Data frame 4. Name of the attribute whose values are to be returned and output - values of an attribute in parameter 4, for the records that match the selection criteria .An **overloaded function** of this would return a filtered list of values from specified column input with input 1. Data frame 2. Name of the attribute whose values are to be returned and output - values of the attribute in parameter 2, for the records that match the selection criteria

- **c.** **get_records** function returns a filtered set of records and takes as input 1. A variable which has the column to be matched with value 2. Value of the selection criteria 3. Dataframe and output - the records that match the selection criteria
- **d.** A **get_count_dictionary** method returns a dictionary with the item as keys and the no of occurrences in the list as values.
- **e.** **The get_flat_list** function returns the flattened list of the input list.

Module – plot-functions Class – plot_functions

- **f.** **The plot_hist** method takes as input the dictionary whose keys are plotted along the x-axis and values along the y-axis

Module – module 2 Class – task2a

- **g.** **dict_view_by_country -** method generates dictionary which has the data for views by country of the document
- **h.** **view_by_country** -method plots View by country using the plot_hist function from plot_functions module

Module – module 2 Class – task2b

- **i.** **view_by_continent** – method maps countries to the continent, group by continent and plot the views by continent

Module – module 3 Class – task3a

- **j.** **the view_by_user_agent** method creates a dictionary which has the data for views by user_agent of the document and plots the respective histogram

Module – module 3 Class – task3b

- **k.** **the view_by_browser** method uses a regular expression to extract the browser name from user agent and creates a dictionary which has the data for views by the browser of the document and plots the respective histogram
- **l.** **browser –** method retrieves the browser names from user_agent using a regular expression

Module – module 4 Class -task4

- **m.** **get_readers** - Function that returns all visitor UUIDs of readers of a document. input : document id , output : list of visitors that read the document
- **n.** **get_read_documents** - Function that returns all document UUIDs read by a visitor. input : visitor_uuid, output : list of documents read by the visitor
- **o.** **get_also_likes** - Function that returns all document UUIDs read by a visitor .input : visitor_uuid, output : list of documents read by the visitor
- **p.** **sort_no_of_readers -** Function to create a dictionary of the form doc_id : no_of_readers step 1 : create a flat list of the direct values, step 2 : count the number of occurrences of each unique values

q. **get_also_likes_sorted** - Function that returns all document UUIDs read by a visitor sort by the sort function passed, input : visitor_uuid , sort function output : list of documents read by the visitor in the sorted order

r. **get_also_likes_sorted_top10 –** Gets the most popular the top 10 documents

Class – task5

s. **also_like_graph :** Function to generate the digraph of also_likes documents

t. **edge_dict** : Function returns a list of the form (reader,doc_id) for all documents read by the reader. This is used to draw the edges of the also_likes digraph.

- **Module – module 6**

  Class – task6

a. **fn**_2a – will get the dictionary for plotting histogram from view_by_dictionary of task2a. and plots it by creates a canvas in the tkinter frame.

Data Structures

- **Dictionary** – To store the details required for plotting histogram
- **List** – To store the intermediate results
- **Data frames –** To store the input data from the file

Libraries

- **Pandas** – for handling the data frames
- **Matplotlib –** For plotting Histogram
- **Numpy** – For manipulating arrays
- **Tkinter** – For GUI
- **Graphviz** – To plot digraph of also_likes functionality
- **Re** – To extract browser names from visitor_useragent
- **Functions** – For higher order functions like reduce

Advanced Language Constructs

The application uses the following advanced concepts in C#.

- **Higher order functions -map,reduce – map** function was used to apply the same function to all the elements of a list.**reduce** function was used to flatten a list having another list as its element.

- **Dictionary Comprehension** – was used to create dictionaries storing the details for plotting
- **List comprehension** – was used to create lists to store intermediate data
- **Classes- Method Overloading** – overloaded methods get_values will fetch the value of a selected column for the records that match the selection criteria in one version whereas the other version will fetch the full record for the instances that match the selection criteria
- **Lambda expressions –** used within higher-order functions **t**o make the function clear and concise
- **Regular Expressions** – **re.findall()** method was used to extract multiple browser names from a single visitor_useragent.

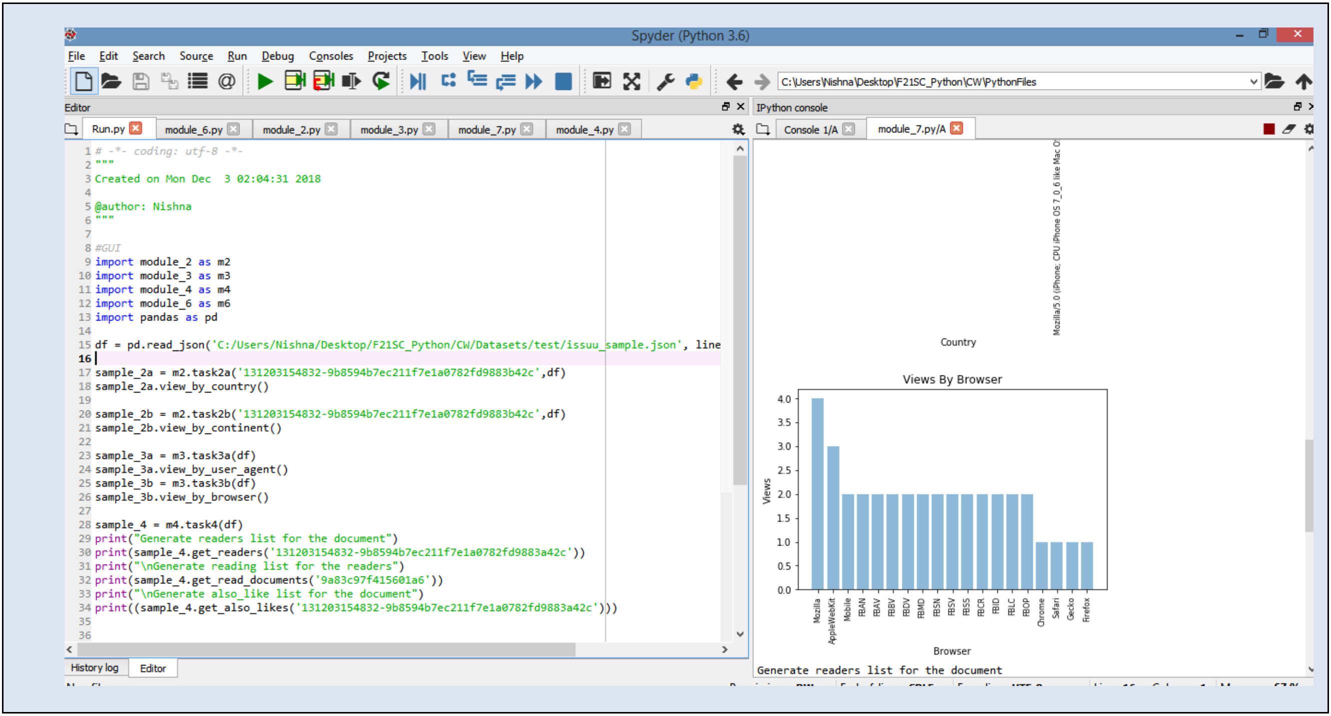
Programming Paradigm

A combination of the following paradigms are used

- **OOP –** Lets you do a modular programming approach
- **Functional Programming –**
    - **a.** Functions as the first-class citizens
    - **b.** Recursions instead of iterations
    - **c.** Stateless

User Guide

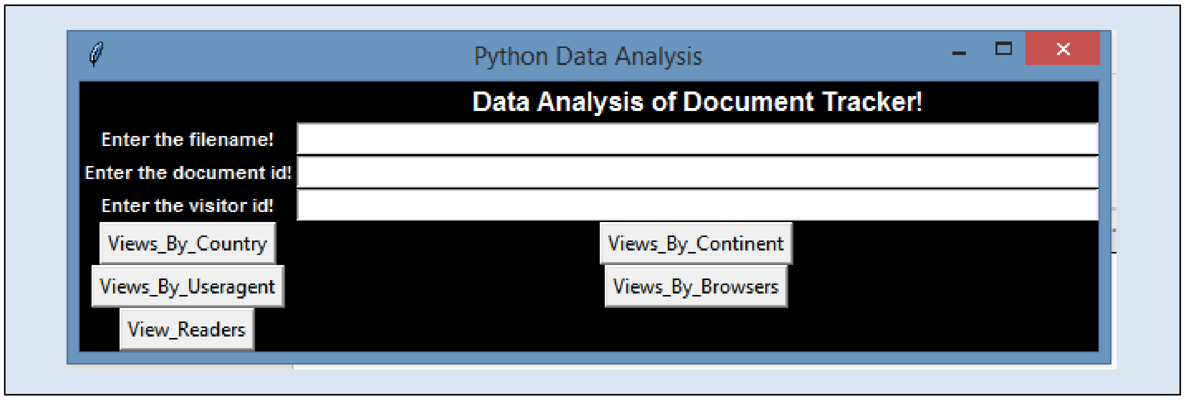
1. The application can be run via GUI or command line tools. Run the run.py module to run the application in the python console



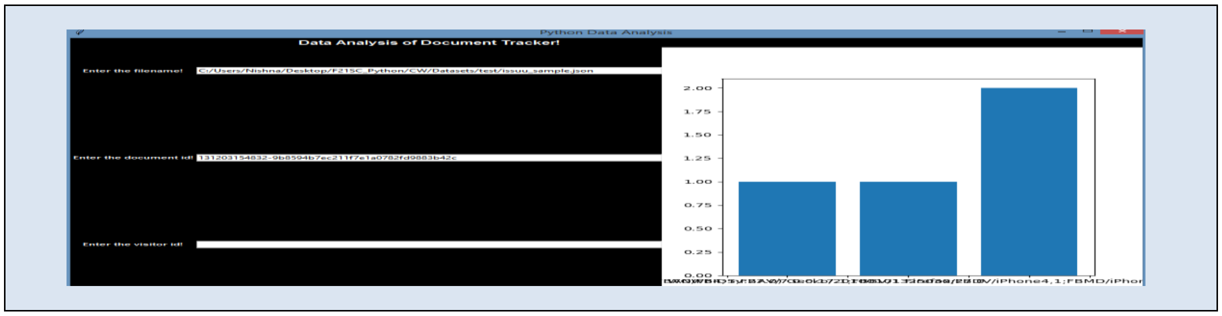
```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  3 02:04:31 2018

@author: Nishna
"""


#GUI
import module_2 as m2
import module_3 as m3
import module_4 as m4
import module_6 as m6
import pandas as pd

df = pd.read_json('C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json', line

sample_2a = m2.task2a('131203154832-9b8594b7ec211f7e1a0782fd9883b42c',df)
sample_2a.view_by_country()

sample_2b = m2.task2b('131203154832-9b8594b7ec211f7e1a0782fd9883b42c',df)
sample_2b.view_by_continent()

sample_3a = m3.task3a(df)
sample_3a.view_by_user_agent()
sample_3b = m3.task3b(df)
sample_3b.view_by_browser()

sample_4 = m4.task4(df)
print("Generate readers list for the document")
print(sample_4.get_readers('131203154832-9b8594b7ec211f7e1a0782fd9883a42c'))
print("\nGenerate reading list for the readers")
print(sample_4.get_read_documents('9a83c97f415601a6'))
print("\nGenerate also_like list for the document")
print((sample_4.get_also_likes('131203154832-9b8594b7ec211f7e1a0782fd9883a42c')))
```

```
Generate readers list for the document
```

2. GUI looks as below



3. Enter the filename and document id and click on the buttons to vieqw the respective plots

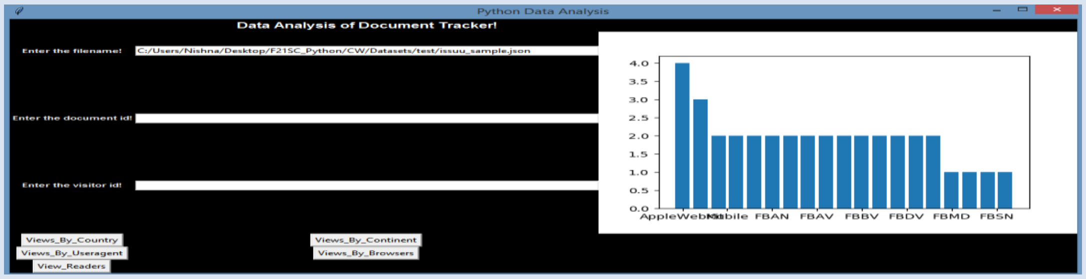


Enter the filename! C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json

Enter the document id! 131203154832-9b8594b7ec211f7e1a0782fd9883b42c

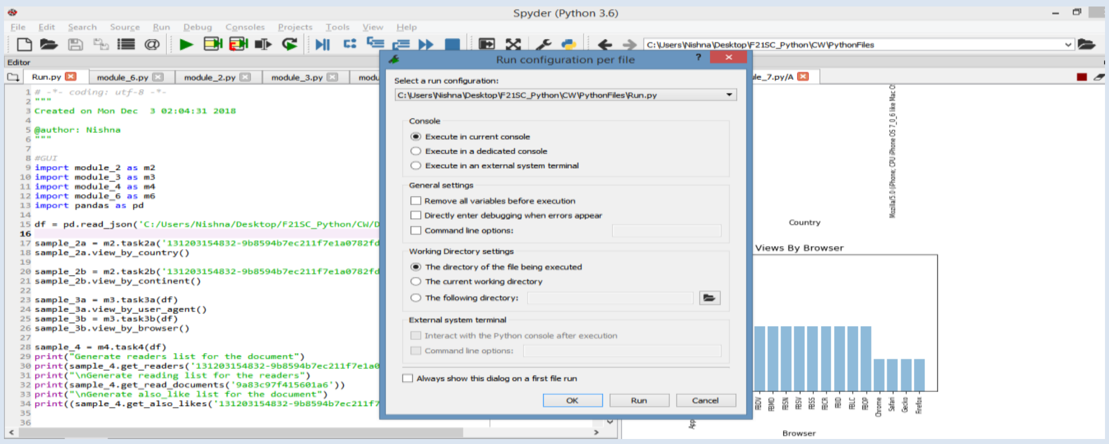
Enter the visitor id!

Nishna



```
Enter the filename! C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json
Enter the document id!
Enter the visitor id!
```

To execute in command line interface, Run -> Configuration per file -> Command Line



```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  3 02:04:31 2018

@author: Nishna
"""

#GUI
import module_2 as m2
import module_3 as m3
import module_4 as m4
import module_6 as m6
import pandas as pd

df = pd.read_json('C:/Users/Nishna/Desktop/F21SC_Python/CW/D

sample_2a = m2.task2a('131203154832-9b8594b7ec211f7e1a0782fd
sample_2a.view_by_country()

sample_2b = m2.task2b('131203154832-9b8594b7ec211f7e1a0782fd
sample_2b.view_by_continent()

sample_3a = m3.task3a(df)
sample_3a.view_by_user_agent()
sample_3b = m3.task3b(df)
sample_3b.view_by_browser()

sample_4 = m4.task4(df)
print("Generate readers list for the document")
print(sample_4.get_readers('131203154832-9b8594b7ec211f7e1a0
print("\nGenerate reading list for the readers")
print(sample_4.get_read_documents('9a83c97f415601a6'))
print("\nGenerate also_like list for the document")
print((sample_4.get_also_likes('131203154832-9b8594b7ec211f7
```

Run configuration per file

Select a run configuration: C:\Users\Nishna\Desktop\F21SC_Python\CW\PythonFiles\Run.py

Console
- [✓] Execute in current console
- Execute in a dedicated console
- Execute in an external system terminal

General settings
- Remove all variables before execution
- Directly enter debugging when errors appear
- Command line options:

Working Directory settings
- [✓] The directory of the file being executed
- The current working directory
- The following directory:

External system terminal
- Interact with the Python console after execution
- Command line options:

Always show this dialog on a first file run

OK Run Cancel

Developer Guide

Key Code Snippets

Read_data – The read_json method from pandas library is used to read the JSON data as a pandas dataframe

```
import pandas as pd

pd.read_json(filepath, lines = True, orient='columns')
```

Filter Columns – To get the values of a particular column for the records that match a selected criteria.

```
def get_values(data, output_attribute, input_attribute= None, value = None):

        return (data[data[input_attribute] == value][output_attribute])
```

Creating a dictionary of counts- This is achieved using dictionary comprehension

```
def get_count_dictionary(input_list) :

""" Function that returns a dictionary with the item as keys and the no of occurrences
in the list as values"""

        return {x : input_list.count(x) for x in input_list}
```

Flaten a list having lists as its element - Use the higher order function reduce from the library functools in combination with lambda expression

```
   import functools as ft
   def get_flat_list(input_list) :
           """ Function that returns the flattened list"""

           return ft.reduce(lambda x,y: x + y,input_list)
```

Plot histogram from a dictionary

```
def plot_hist(dict, xlab, ylab, title): # plot histogram
        import matplotlib.pyplot as plt
        import numpy as np

        """extracting values"""
        y = dict.keys()
        x = dict.values()
        """plotting"""
        plt.bar(y_pos, x, align='center', alpha=0.5)

           plt.xticks(y_pos, y, fontsize=8, rotation='vertical')           }
```

Create dictionary for view by continent – First, get the list of records., add a new Column – Continents, group by continent & get view count . create dictionary which has the data for views by country of the document

```
records = cf.common_functions.get_values(self.__datafile,
                                                 'visitor_country',
                                                 'subject_doc_id',
                                                 self.__document_id).to_frame() #
        records['visitor_continent'] =
records['visitor_country'].map(self.__country_to_continent).map(self.__continents)
```

```
        view_count = records.groupby(['visitor_continent']).size() #Group by
continent & get view count
        country_dict = {view_count.index[i] : view_count[i] for i in
range(0,len(view_count))}
```

View By Browser – Function retreives the browser names from user_agent using regular expression. Get the list from each user agent. Flatten the above list. Dictionary which has the data for views by browser of the document

```
def browser(self,n):

        import re

        return re.findall(r'(\w+)/',n)

browser_lists = map(self.browser,useragents)

browser_flatlist = cf.common_functions.get_flat_list(browser_lists)

country_dict = cf.common_functions.get_count_dictionary(browser_flatlist)
```

Sorting based on number of readers – Function to create dictionary of the form doc_id : no_of_readers step 1 : create a flat list of the dict values step 2 : count the number of occurances of each unique values

```
doc_list = cf.common_functions.get_flat_list(dict_reading.values())

dict_reading_count = cf.common_functions.get_count_dictionary(doc_list)

dict_reading_count_desc                =                sorted(dict_reading_count,
key=dict_reading_count.__getitem__, reverse=True)
```

Also likes – Function that returns all document UUIDs read by a visitor. First Get the readers of the document. Create dictionary of the form reader: readinglist (contains only unique values of the document ids) using dictionary comprehension.

```
def get_also_likes(self, document_id, user_id = None) :
        readers = self.get_readers(document_id)
        dict_reading = {x : list(set(self.get_read_documents(x))) for x in readers}
return (dict_reading)
```

Also Like Graph – Using Digraph from graphviz create a graph. Add square nodes for each reader by traversing through the dictonary keys and circle nodes for the documents by traversing through the dict values.edge dict function will generate a list of the form(key, value) with last 4 characters of the items.

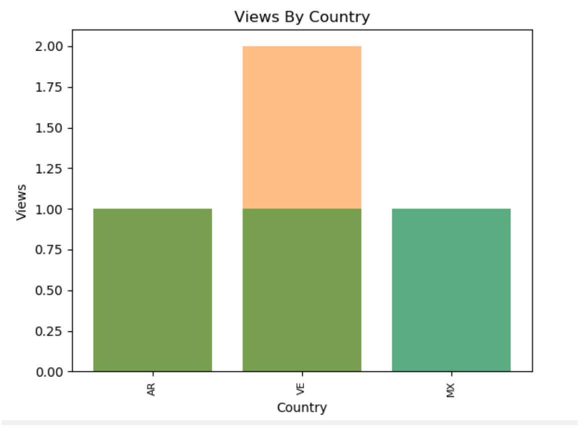
```
from graphviz import Digraph
graph = Digraph('G', filename = 'SpyderGraph.gv').xml");
for k in dict_also_like.keys() :
            graph.node(k[-4:],shape = 'square')
            for edge in task5.edge_dict(self,k,dict_also_like)  :
                graph.edge(edge[0],edge[1], shape = 'square')

def edge_dict(self,key,dict_also_like) :
        return [(key[-4:],v[-4:]) for v in dict_also_like[key]]
```

Plotting histograms in tkinter – Get the details to plot from view_by_continent function in module 2. Extract the values. Create a canvas in tkinter and plot.

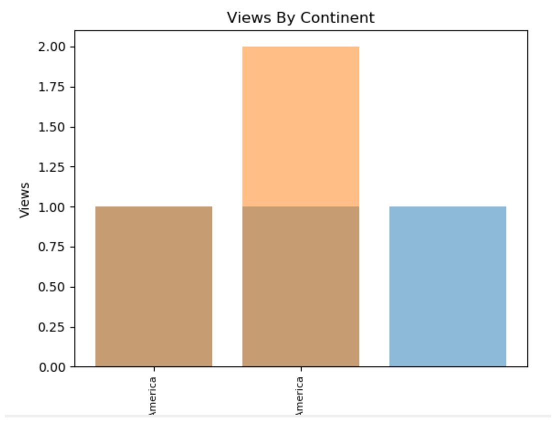
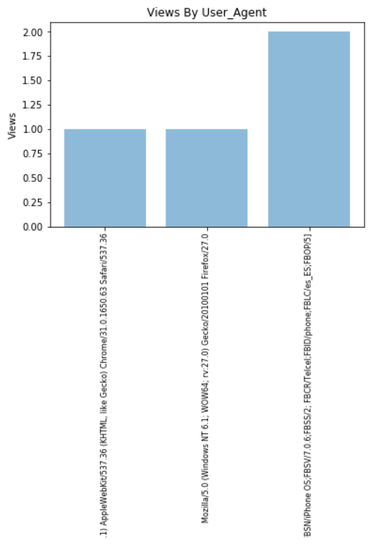
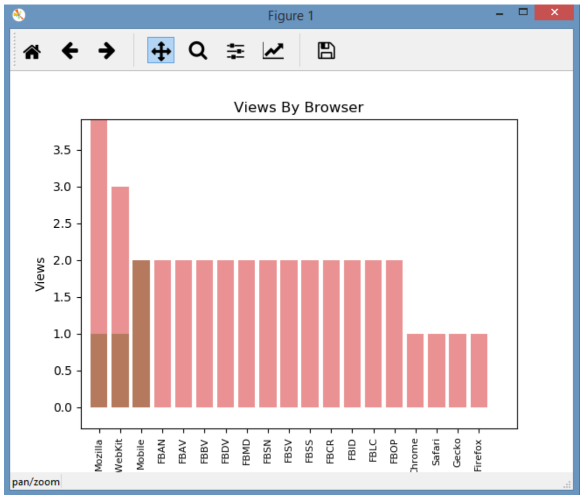
```
dict, xlab, ylab, title = sample_2b.view_by_continent(gui = 'true') #get the
details to plot

        """extracting values"""
        y = dict.keys()
        y_pos = np.arange(len(y))
        x = dict.values()

        """plotting"""
        f = matplotlib.pyplot.Figure(figsize=(5,4), dpi=100)
        ax = f.add_subplot(111)
        ax.bar(y_pos, x)
        ax.set_xticklabels(labels = y)
        canvas = FigureCanvasTkAgg(f)
        canvas.get_tk_widget().grid(row=3, column=3, rowspan=6)

        canvas.show()
```

Testing

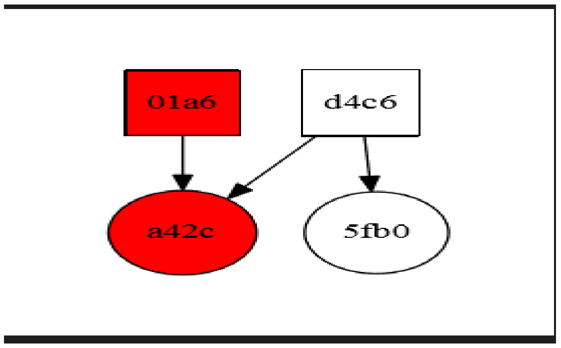
The testing performed on the following functionalities has been successful application are listed below.

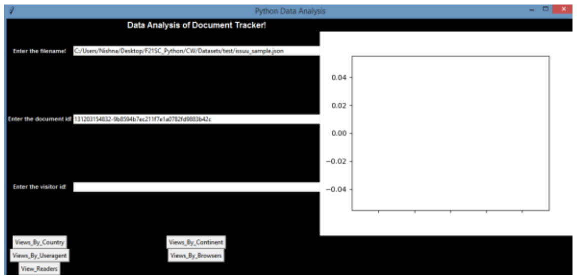
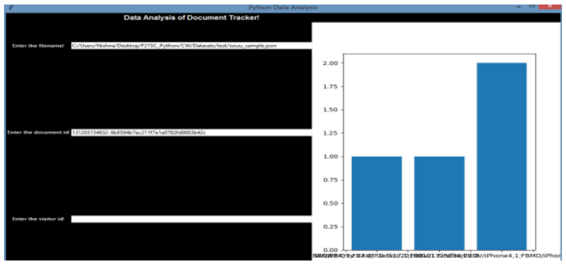
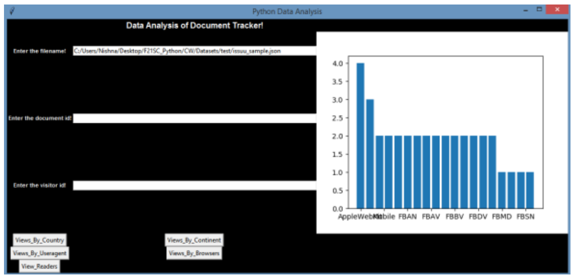
Test Case	Results	Status
View By Country		Success

Nishna

View By Continent		Success
View By User agent		Success
View By Browser	 Figure 1 Views By Browser Views 3.5 3.0 2.5 2.0 1.5 1.0 0.5 0.0 Mozilla WebKit Mobile FBAN FBAV FBBV FBDV FBMD FBSN FBSV FBSS FBCR FBID FBLC FBOP Chrome Safari Gecko Firefox pan/zoom	Success

Nishna

Get Readers	Generate readers list for the document ['9a83c97f415601a6', '745409913574d4c6', '9a83c97f415601a6'] Generate reading list for the readers ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '131203154832-9b8594b7ec211f7e1a0782fd9883a42c'] Generate also_like list for the document {'9a83c97f415601a6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c'], '745409913574d4c6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']} ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']	Success
Get Reading List	Generate readers list for the document ['9a83c97f415601a6', '745409913574d4c6', '9a83c97f415601a6'] Generate reading list for the readers ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '131203154832-9b8594b7ec211f7e1a0782fd9883a42c'] Generate also_like list for the document {'9a83c97f415601a6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c'], '745409913574d4c6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']} ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']	Success
Get also likes	Generate readers list for the document ['9a83c97f415601a6', '745409913574d4c6', '9a83c97f415601a6'] Generate reading list for the readers ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '131203154832-9b8594b7ec211f7e1a0782fd9883a42c'] Generate also_like list for the document {'9a83c97f415601a6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c'], '745409913574d4c6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']} ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']	Success
Get also like sorted	Generate readers list for the document ['9a83c97f415601a6', '745409913574d4c6', '9a83c97f415601a6'] Generate reading list for the readers ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '131203154832-9b8594b7ec211f7e1a0782fd9883a42c'] Generate also_like list for the document {'9a83c97f415601a6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c'], '745409913574d4c6': ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']} Generate also_like top 10 list sorted based on number of readers for the document ['131203154832-9b8594b7ec211f7e1a0782fd9883a42c', '140228202800-6ef39a241f35301a9a42cd0ed21e5fb0']	Success
Also likes graph		Success

View By Country – Using GUI	 Python Data Analysis Data Analysis of Document Tracker! Enter the filename! C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json Enter the document id! 131203154832-9b8594b7ec211f7e1a0782fd9883b42c Enter the visitor id!	Success
View By Continent – Using GUI		Success
View By User agent – Using GUI	 Python Data Analysis Data Analysis of Document Tracker! Enter the filename! C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json Enter the document id! 131203154832-9b8594b7ec211f7e1a0782fd9883b42c Enter the visitor id!	Success
View By Browser – Using GUI	 Python Data Analysis Data Analysis of Document Tracker! Enter the filename! C:/Users/Nishna/Desktop/F21SC_Python/CW/Datasets/test/issuu_sample.json Enter the document id! Enter the visitor id!	Success

Reflection

- While developing this application, lots of advanced features from core Python and the Libraries puts the development to ease. They include
    - Higher order functions -map,reduce
    - Dictionary Comprehension,
    - List comprehension
    - Classes- Method Overloading
    - Lambda expressions within higher order functions

  - Regular Expressions
  - Design Patterns
- Limitations of this application and suggested solutions
  - **a.** Also like graph could not be displayed in GUI
    - Solution – Import graph as an image from pdf and display it in GUI

Conclusion

A Data Analysis of a Document Tracker application with the detailed requirement checklist has been developed. The Data Analysis of a Document Tracker application development opened the opportunity to learn advanced concepts in Python like Higher order functions, dictionary comprehension, regular expressions, list comprehensions, lambda expressions and so on.

References

Boyanov, A. (2018). Python Design Patterns: For Sleek And Fashionable Code. [Blog] total. Available at: https://www.toptal.com/python/python-design-patterns [Accessed 7 Dec. 2018].

Sourcemaking.com. (2018). *Design Patterns and Refactoring*. [online] Available at: https://sourcemaking.com/design_patterns/chain_of_responsibility [Accessed 7 Dec. 2018].